FIG. 1

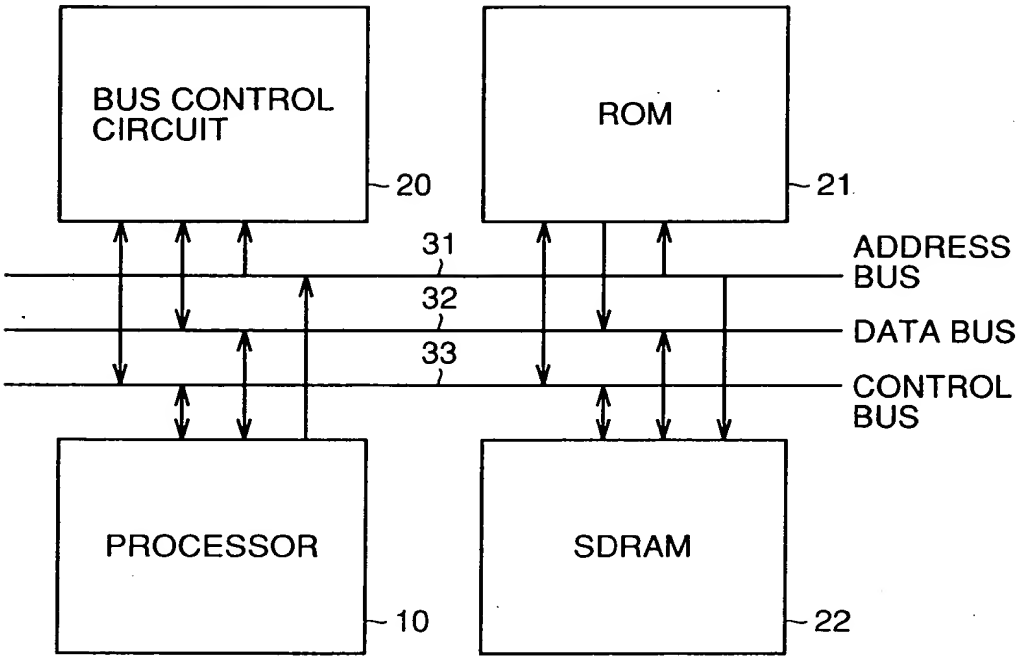


FIG.2

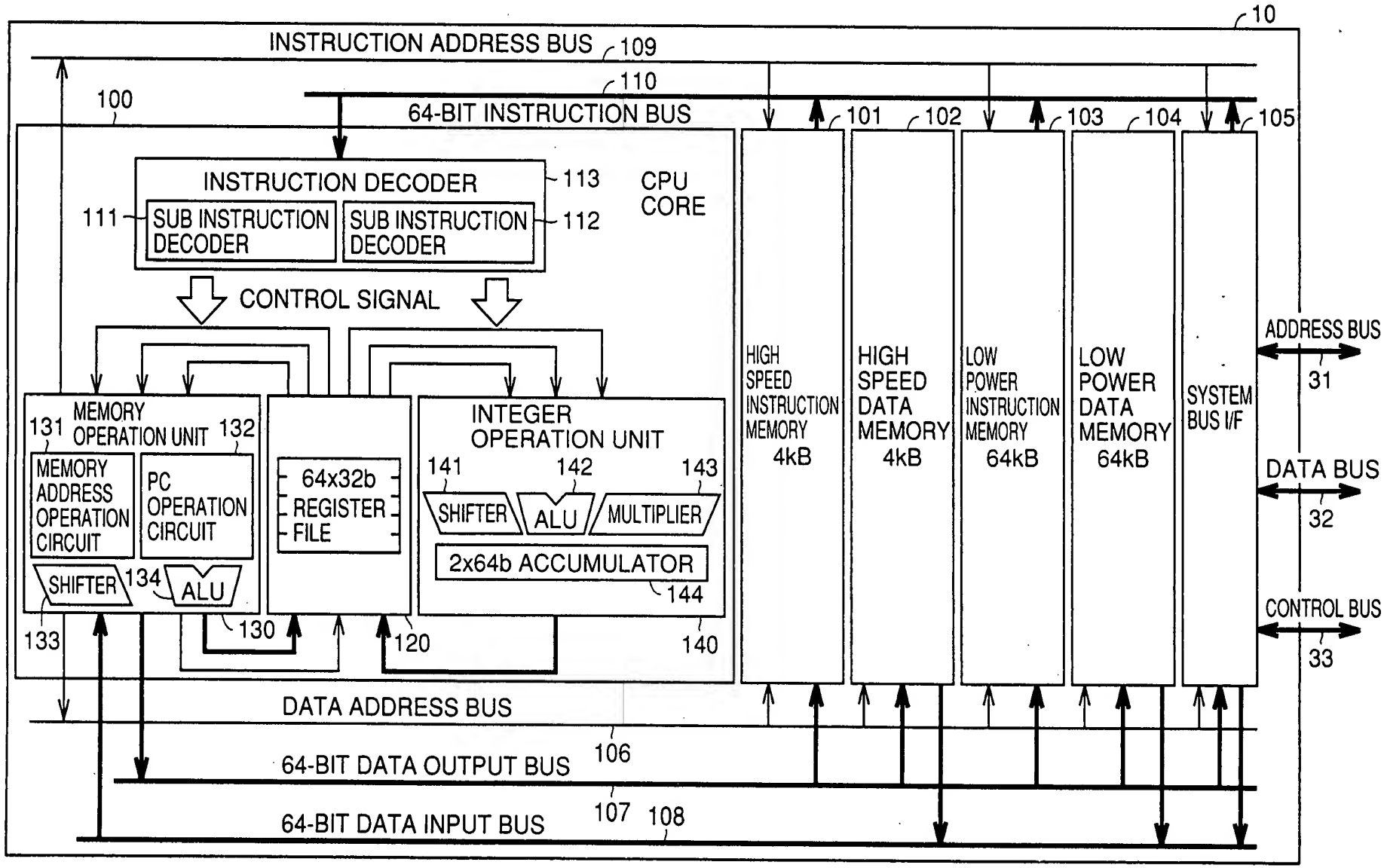


FIG.3

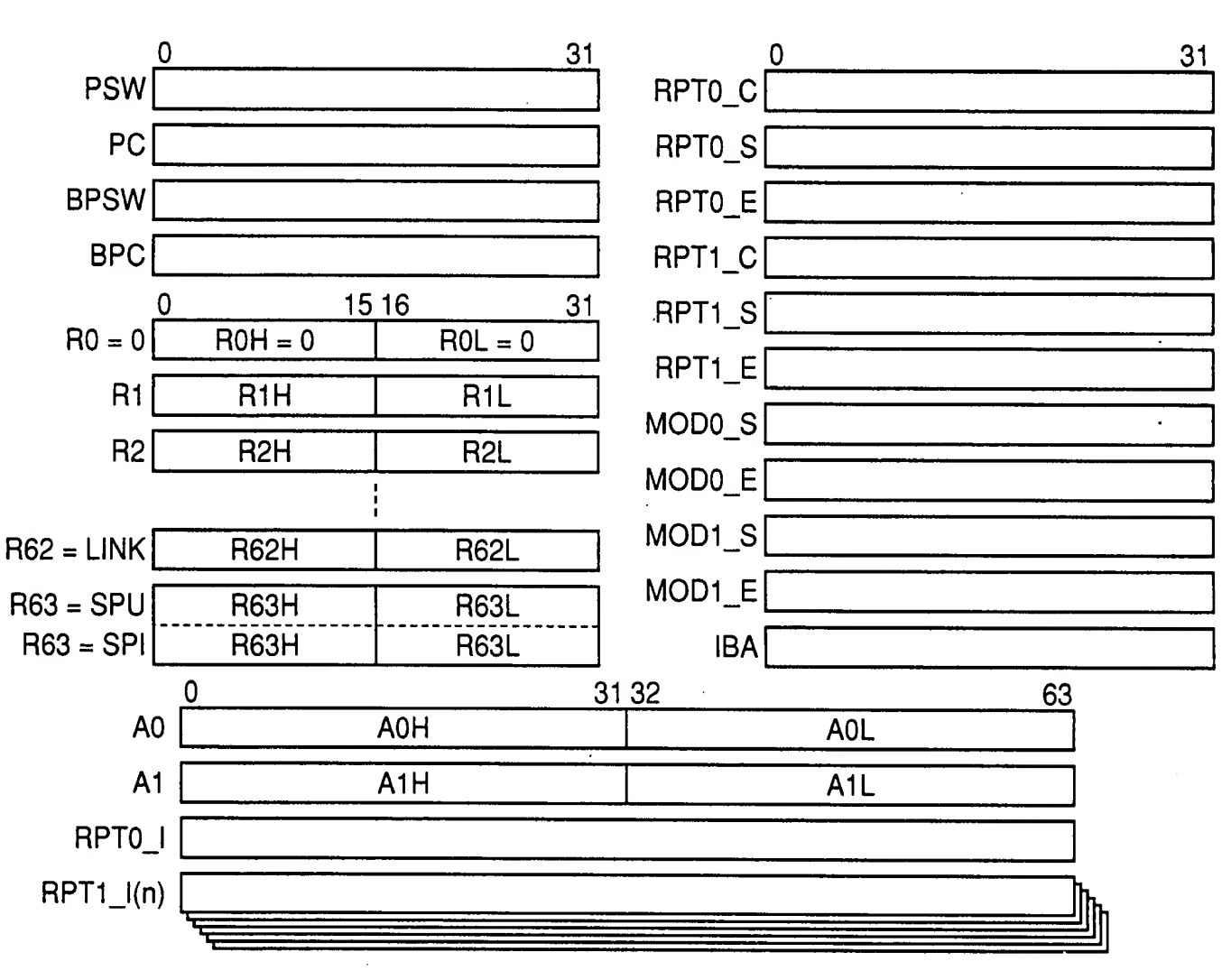


FIG.4

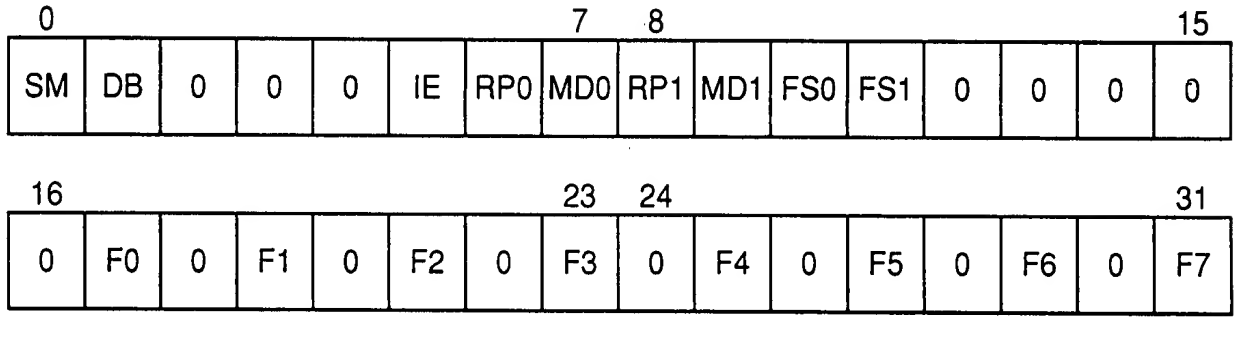
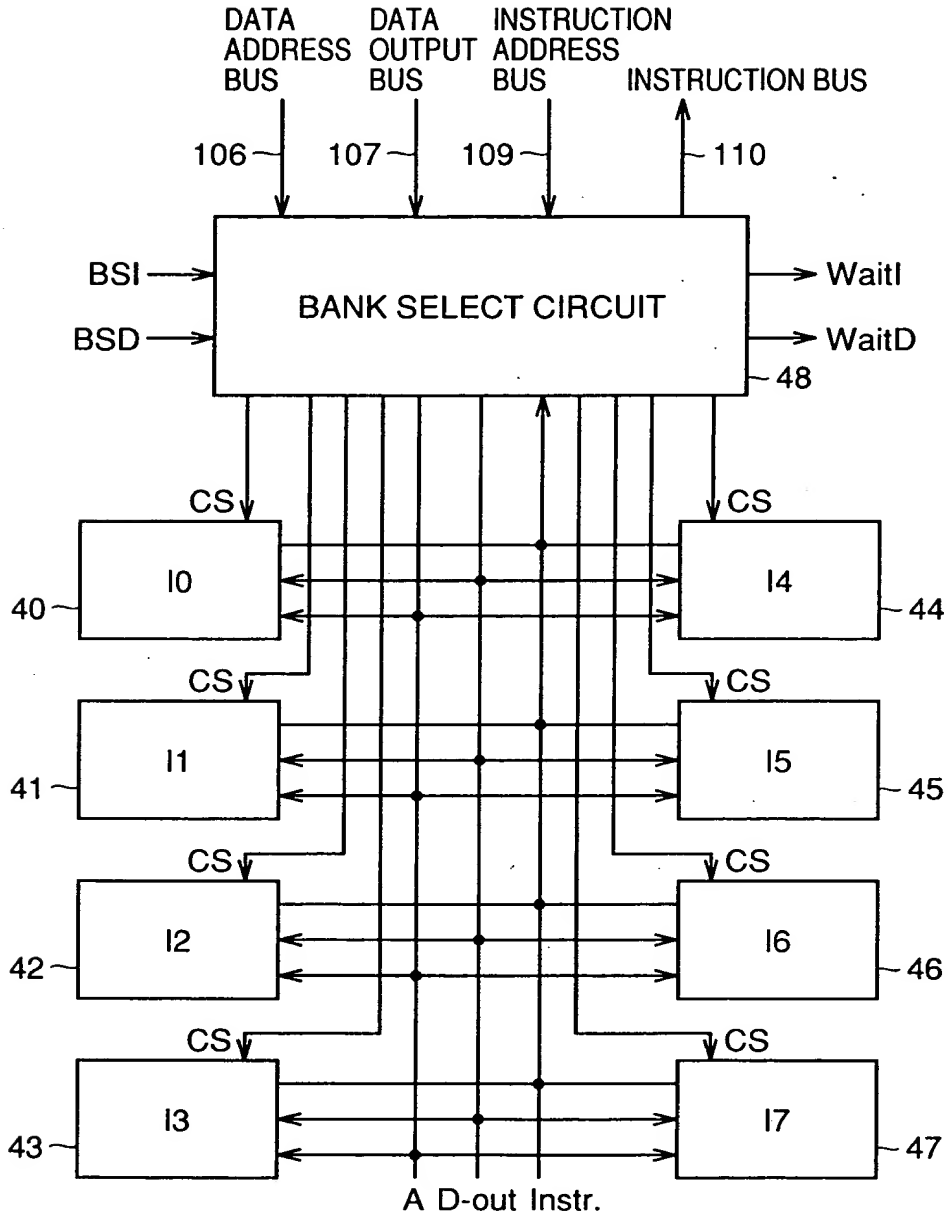


FIG.5


DATA ADDRESS BUS
DATA OUTPUT BUS
INSTRUCTION ADDRESS BUS
INSTRUCTION BUS
106
107
109
110
BSI
BSD
BANK SELECT CIRCUIT
WaitI
WaitD
48
CS
I0
I1
I2
I3
I4
I5
I6
I7
40
41
42
43
44
45
46
47
A D-out Instr.

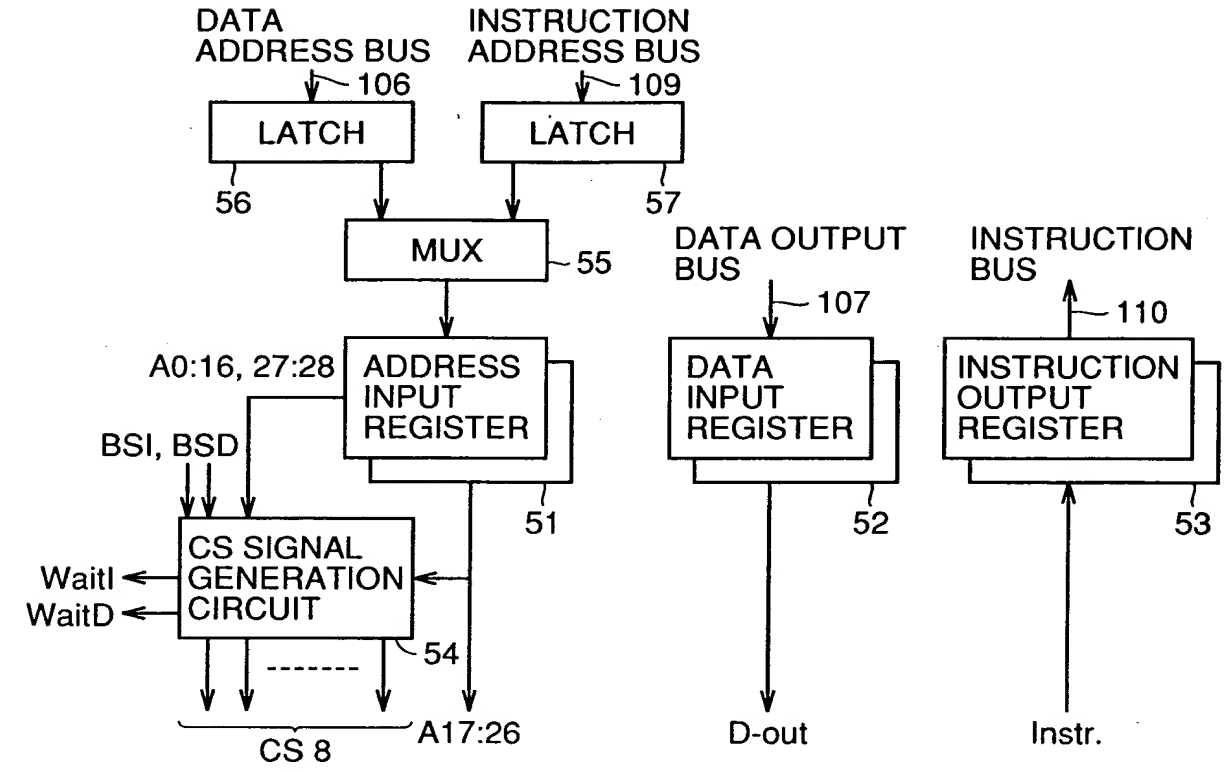


FIG.6

FIG.7

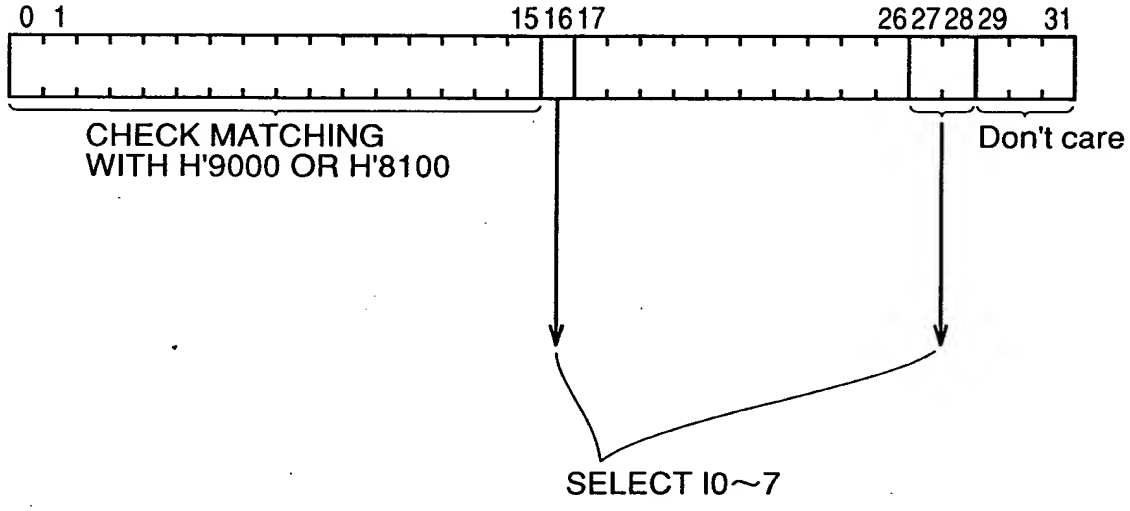


FIG.8

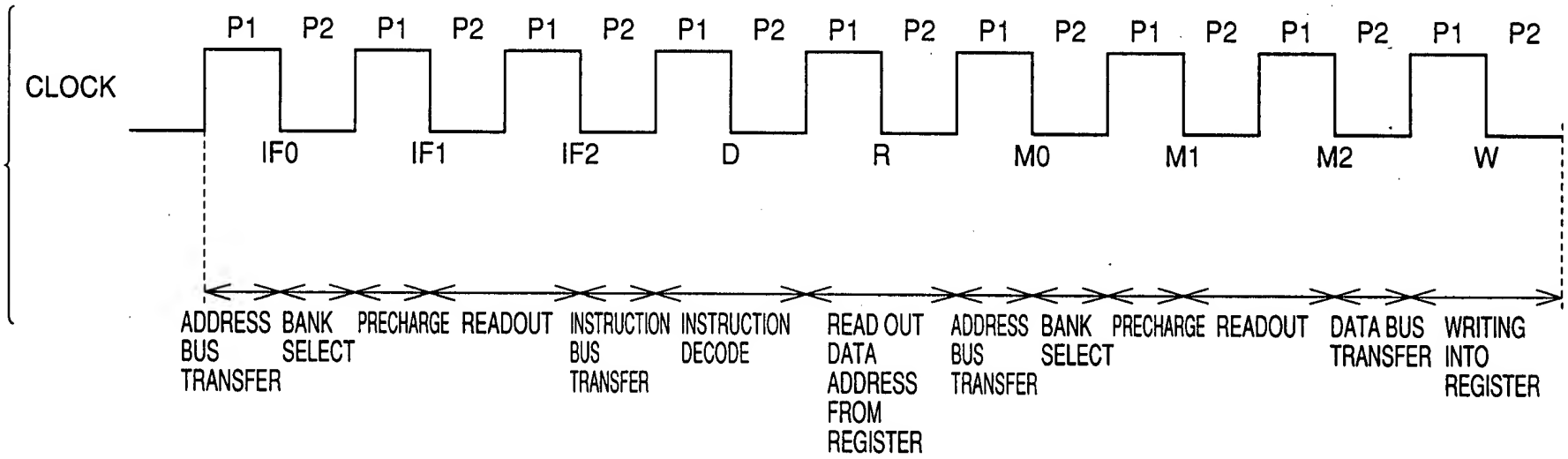
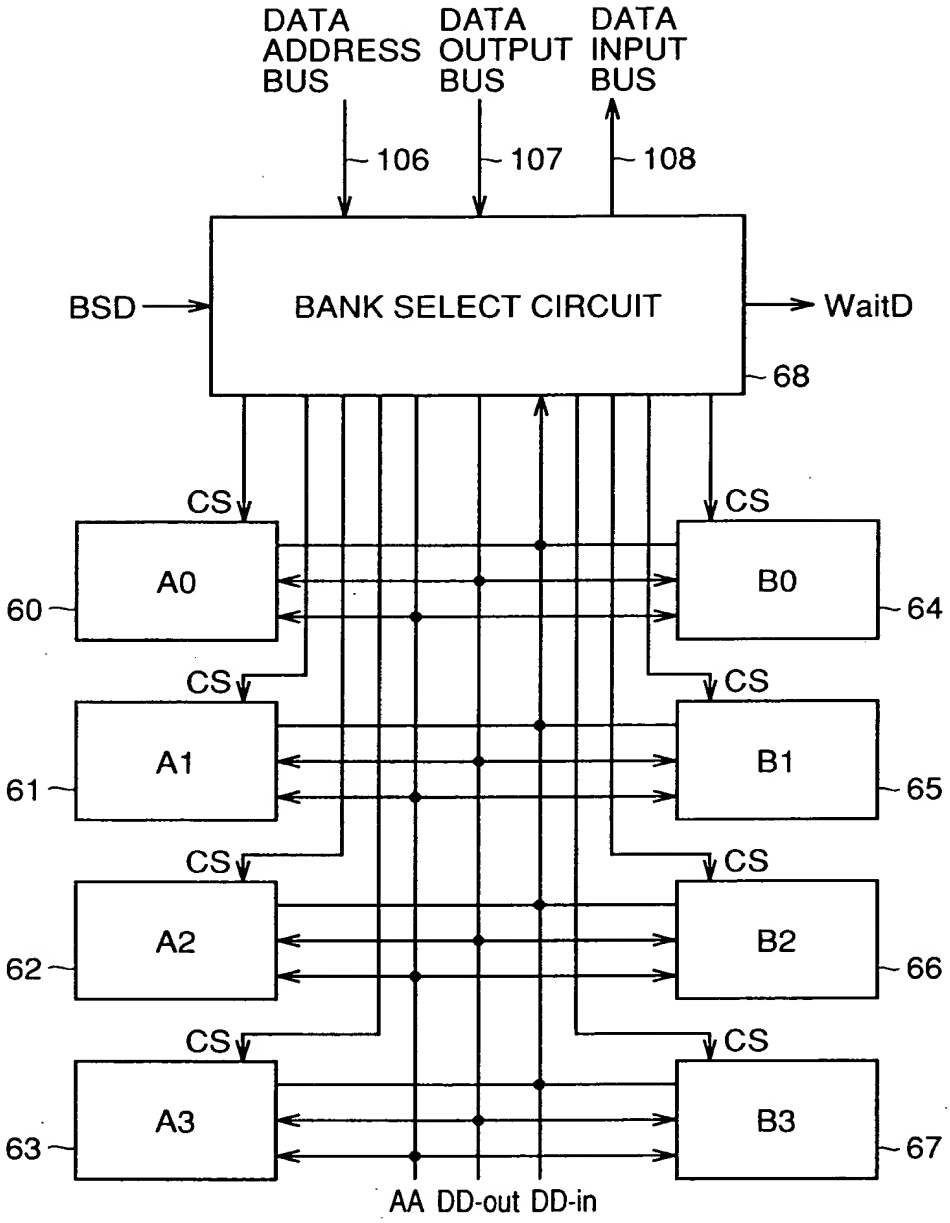


FIG.9



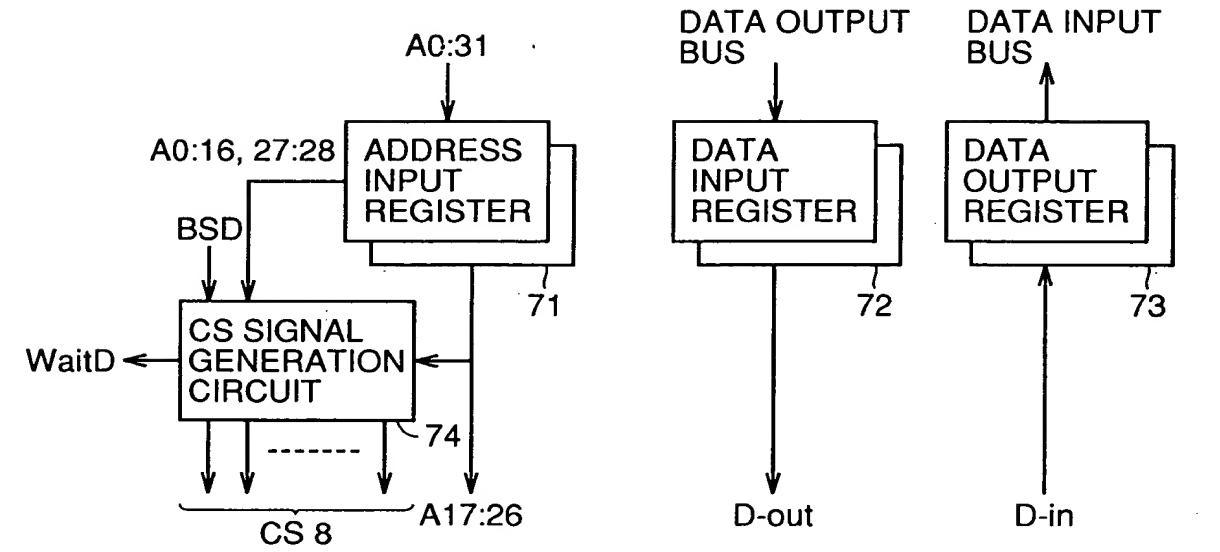


FIG.10

FIG.11

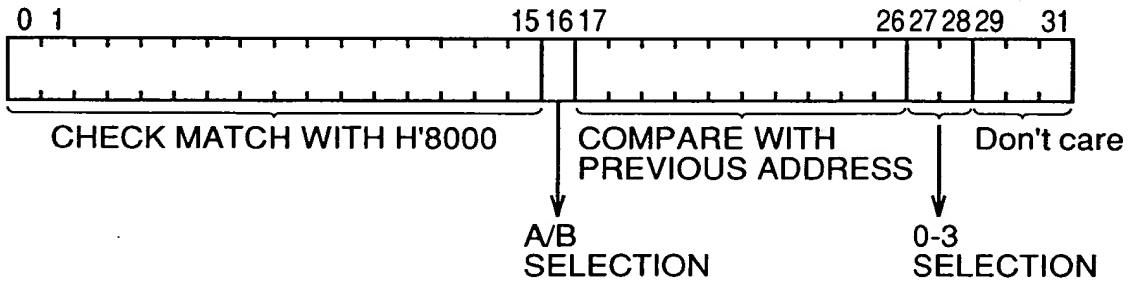


FIG. 12A

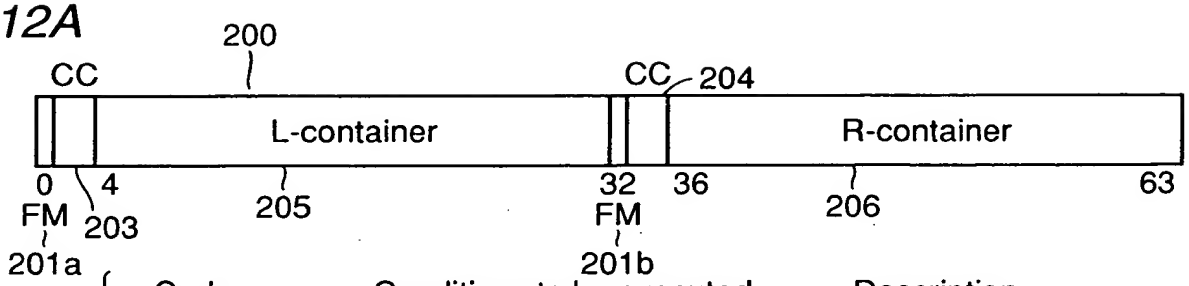


FIG. 12B

• Code	Conditions to be executed	Description
CC = 000	Always	---
CC = 001	F0 = T and F1 = don't care	/TX
CC = 010	F0 = F and F1 = don't care	/FX
CC = 011	F0 = don't care and F1 = T	/XT
CC = 100	F0 = don't care and F1 = F	/XF
CC = 101	F0 = T and F1 = T	/TT
CC = 110	F0 = T and F1 = F	/TF
CC = 111	Reserved	---

FIG. 12C

FM	Number of sub-instructions	Issuing Order	
		L-container	R-container
00	two	1st	1st
01	two	1st	2nd
10	two	2nd	1st
11	one	1st	----

FIG.13A Short_M

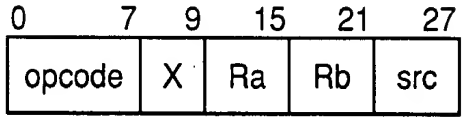


FIG.13B Short_A

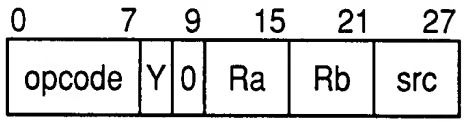


FIG.13C Short_B1

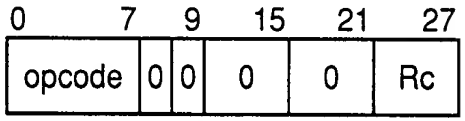


FIG.13D Short_B2

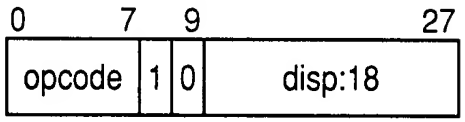


FIG.13E Short_B3

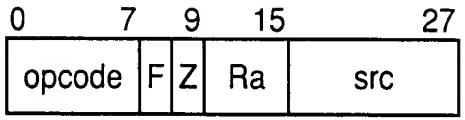


FIG.13F Short_D1

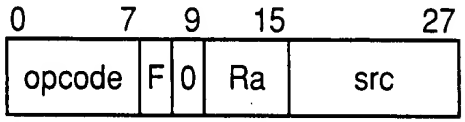


FIG.13G Short_D2

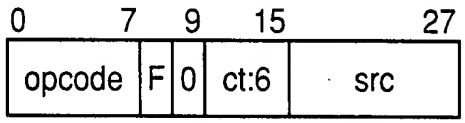


FIG.13H Long

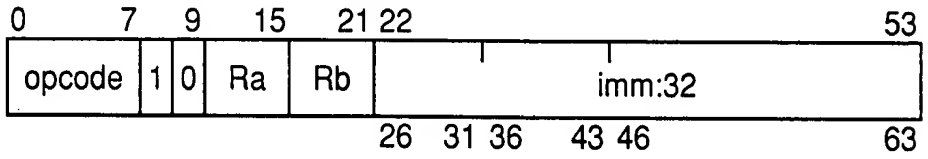


FIG.14A

FIG.14B

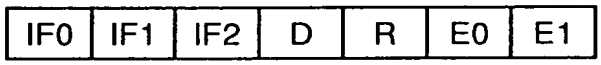


FIG.14C

FIG.14D

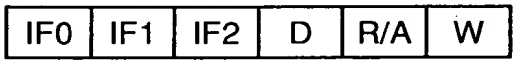


FIG.14E

FIG.14F

FIG.14G

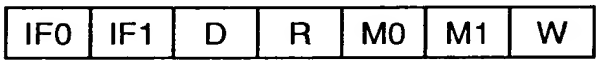


FIG.14H

FIG.15

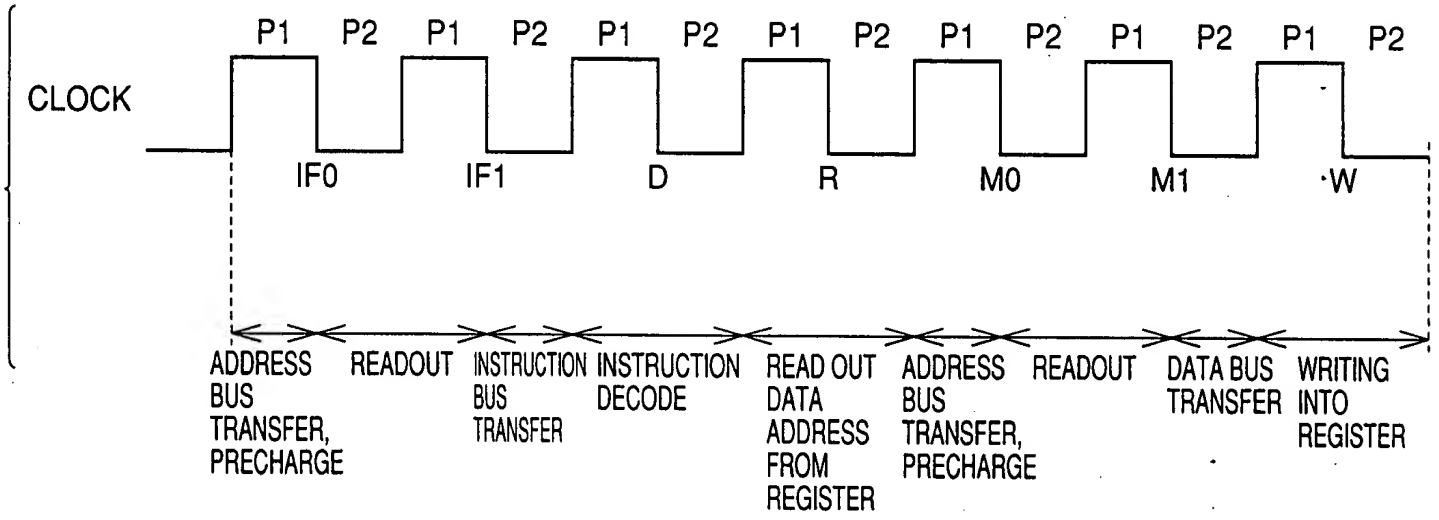

CLOCK
P1
P2
P1
P2
P1
P2
P1
P2
P1
P2
P1
P2
P1
P2
IF0
IF1
D
R
M0
M1
W
ADDRESS BUS TRANSFER, PRECHARGE
READOUT
INSTRUCTION BUS TRANSFER
INSTRUCTION DECODE
READ OUT DATA ADDRESS FROM REGISTER
ADDRESS BUS TRANSFER, PRECHARGE
READOUT
DATA BUS TRANSFER
WRITING INTO REGISTER

FIG.16

- Load/Store instructions

LDB	Load one byte to a register with sign extension
LDBU	Load one byte to a register with zero extension
LDH	Load one half-word to a register with sign extension
LDHH	Load one half-word to a register high
LDHU	Load one half-word to a register with zero extension
LDW	Load one word to a register
LD2W	Load two words to registers
LD4BH	Load four bytes to four half-word registers with sign extension
LD4BHU	Load four bytes to four half-word registers with zero extension
LD2H	Load two half-words to registers
STB	Store one byte from a register
STH	Store one half-word from a register
STHH	Store one half-word from a register high
STW	Store one word from a register
ST2W	Store two words from registers
ST4HB	Store four bytes from four half-word registers
ST2H	Store two half-words from registers
MODDEC	Decrement a register value by a 5-bit immediate value
MODINC	Increment a register value by a 5-bit immediate value

- Transfer instructions

MVFSYS	Move a control register to a general purpose register
MVTSYS	Move a general purpose register to a control register
MVFACC	Move a word from an accumulator
MVTACC	Move two general purpose registers to an accumulator

- Compare instructions

CMPcc	Compare cc = EQ (000), NE (001), GT (010), GE (011), LT (100), LE (101), PS - both positive (110), NG - both negative (111)
CMPUcc	Compare unsigned cc = GT (010), GE (011), LT (100), LE (101)

FIG. 17

- Arithmetic operation instructions

ABS	Absolute
ADD	Add
ADDC	Add with carry
ADDHppp	Add half-word
	ppp = LLL (000), LLH (001), LHL (010), LHH (011), HLL (100), HLH (101), HHL (110), HHH (111)
ADDS	Add register Rb with the sign of the third operand
ADDS2H	Add sign to two half-word
ADD2H	Add two pairs of half-words
AVG	Average with rounding towards positive infinity
AVG2H	Average two pairs of half-words rounding towards positive infinity
JOINpp	Join two half-words
	pp = LL (00), LH (01), HL (10), HH (11)
SUB	Subtract
SUBB	Subtract with borrow
SUBHppp	Subtract half-word
	ppp = LLL (000), LLH (001), LHL (010), LHH (011), HLL (100), HLH (101), HHL (110), HHH (111)
SUB2H	Subtract two pairs of half-words

- Logical operation instructions

AND	logical AND
OR	logical OR
NOT	logical NOT
XOR	logical exclusive OR
ANDFG	logical AND flags
ORFG	logical OR flags
NOTFG	logical NOT a flag
XORFG	logical exclusive OR flags

- Shift operation instructions

SRA	Shift right arithmetic	
SRAHp	Shift right arithmetic a half-word	p = L (0), H (1)
SRA2H	Shift right arithmetic two half-words	
SRC	Shift right concatenated registers	
SRL	Shift right logical	
SRLHp	Shift right logical a half-word	p = L (0), H (1)
SRL2H	Shift right logical two half-words	
ROT	Rotate right	
ROT2H	Rotate right two half-words	

- Bit operation instructions

BCLR	Clear a bit
BNOT	Invert a bit
BSET	Set a bit
BTST	Test a bit

FIG. 18

- Branch instructions

BRA	Branch
BRATZR	Branch if zero
BRATNZ	Branch if not zero
BSR	Branch to subroutine
BSRTZR	Branch to subroutine if zero
BSRTNZ	Branch to subroutine if not zero
DBRA	Delayed Branch
DBRAI	Delayed Branch immediate
DBSR	Delayed Branch to subroutine
DBSRI	Delayed Branch immediate to subroutine
DJMP	Delayed Jump
DJMPI	Delayed Jump immediate
DJSR	Delayed Jump to subroutine
DJSRI	Delayed Jump immediate to subroutine
JMP	Jump
JMPTZR	Jump if zero
JMPTNZ	Jump if not zero
JSR	Jump to subroutine
JSRTZR	Jump to subroutine if zero
JSRTNZ	Jump to subroutine if not zero
NOP	No operation

- OS-related instructions

TRAP	Trap
REIT	Return from exception, interrupts, and traps

- DSP Arithmetic operation instructions

MUL	Multiply	
MULX	Multiply with extended precision	
MULXS	Multiply and shift to the left by one with extended precision	
MULX2H	Multiply two pairs of half-words with extended precision	
MULHXpp	Multiply two half-words with extended precision pp = LL (00), LH (01), HL (10), HH (11)	
MUL2H	Multiply two pairs of half-words	
MACd	Multiply and add	(d = 0, 1)
MACSd	Multiply, shift to the left by one, and add	(d = 0, 1)
MSUBd	Multiply and subtract	(d = 0, 1)
MSUBSd	Multiply, shift to the left by one, and subtract	(d = 0, 1)
SAT	Saturate	
SATHH	Saturate word operand into high half-word	
SATHL	Saturate word operand into low half-word	
SATZ	Saturate into positive number	
SATZ2H	Saturate two half-words into positive number	
SAT2H	Saturate two half-word operands	

- Repeat instructions

REPEAT0	Repeat a block of instructions #0
REPEAT1	Repeat a block of instructions #1

- Debugger supporting instructions

DBT	Debug trap
RTD	Return from debug interrupt and trap

FIG.19

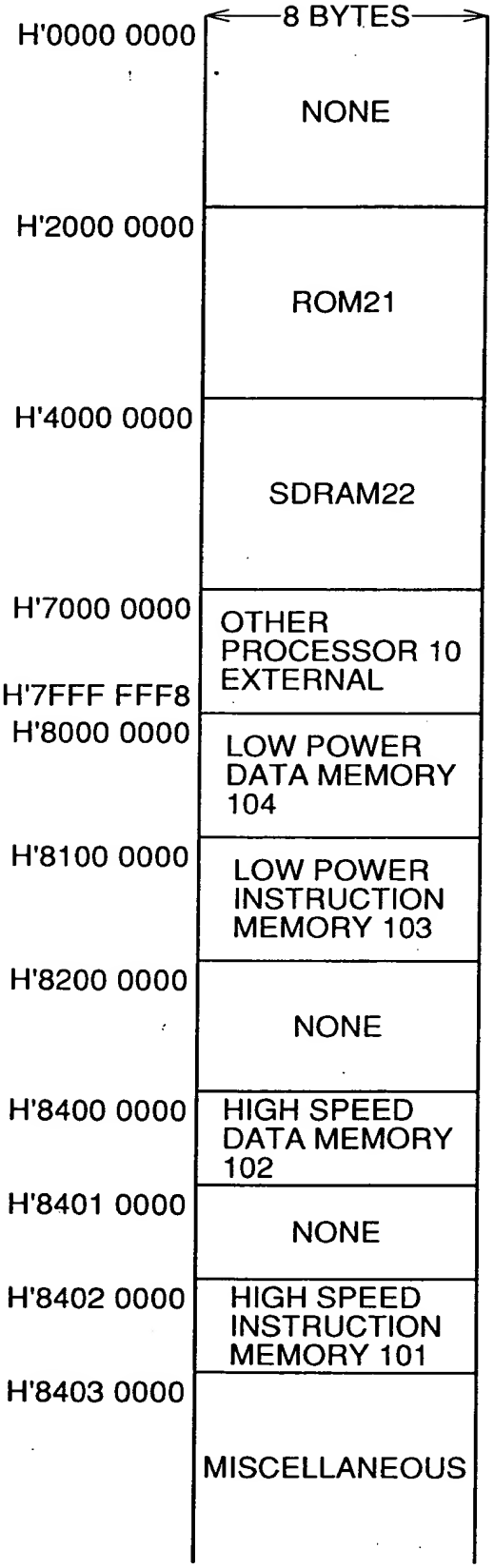


FIG.20

```
• Operation:
        REPEAT1 #count, #pcaddr
                RPT1_C = #count-1
                RPT1_S = PC + 8
                RPT1_E = PC + pcaddr
                RPT1_I(0:5) = Instructions at memory((PC+8):(PC+48))
                if (PC ==RPT1_E && RPT1_C > 0) {
                  RPT1_C--
                  PC == RPT1_S
                }
• Example:
        REPEAT1 #20, #48
START:LD2W R10, @(R30+, R0) ‖ MAC0 R0, R12, R22
        LD2W R20, @(R31+, R0) ‖ MAC0 R0, R13, R23
        LD2W R12, @(R30+, R0) ‖ MAC0 R0, R14, R24
        LD2W R22, @(R31+, R0) ‖ MAC0 R0, R15, R25
        LD2W R14, @(R30+, R0) ‖ MAC0 R0, R16, R26
  END:LD2W R24, @(R31+, R0) ‖ MAC0 R0, R17, R27
```

Zero-overhead loop

FIG.21

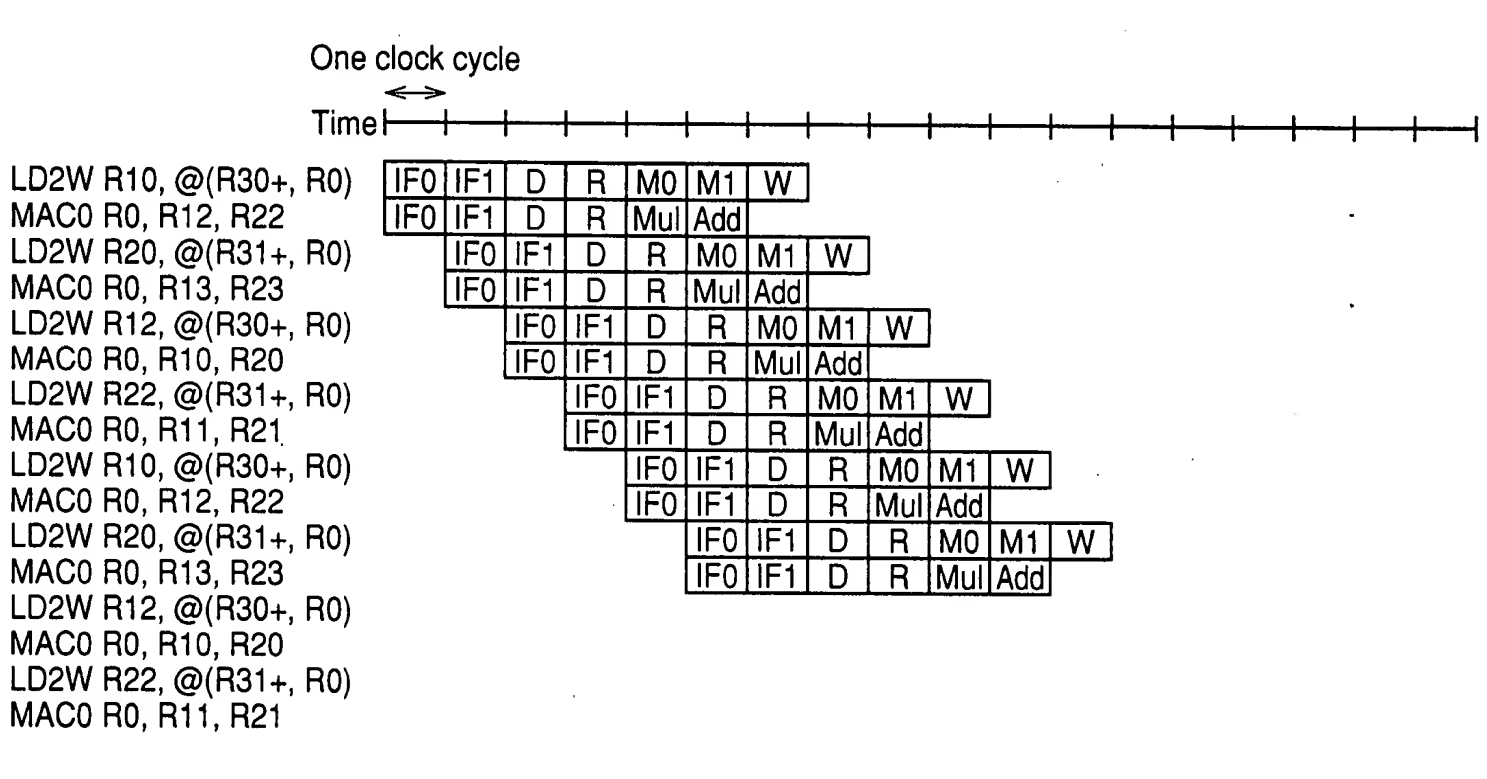


FIG.22

- Operation:

```
REPEAT0 #count, #pcaddr
        RPT0_C = #count-1
        RPT0_S = PC + 8
        RPT0_E = PC + pcaddr
        RPT0_I = Instruction at memory(PC+8)
        if (PC ==RPT0_E && RPT0_C > 0) {
          RPT0_C--
          PC == RPT0_S
        }
```

- Example:

```
        REPEAT0 #10, #64
START:LD2W R10, @(R30+, R0) ‖ MAC0 R0, R12, R22
      LD2W R20, @(R31+, R0) ‖ MAC0 R0, R13, R23
      LD2W R12, @(R30+, R0) ‖ MAC0 R0, R14, R24
      LD2W R22, @(R31+, R0) ‖ MAC0 R0, R15, R25
      LD2W R14, @(R30+, R0) ‖ MAC0 R0, R16, R26
      LD2W R24, @(R31+, R0) ‖ MAC0 R0, R17, R27
      LD2W R16, @(R30+, R0) ‖ MAC0 R0, R10, R20
  END:LD2W R26, @(R31+, R0) ‖ MAC0 R0, R11, R21
```

Zero-overhead loop

FIG.23

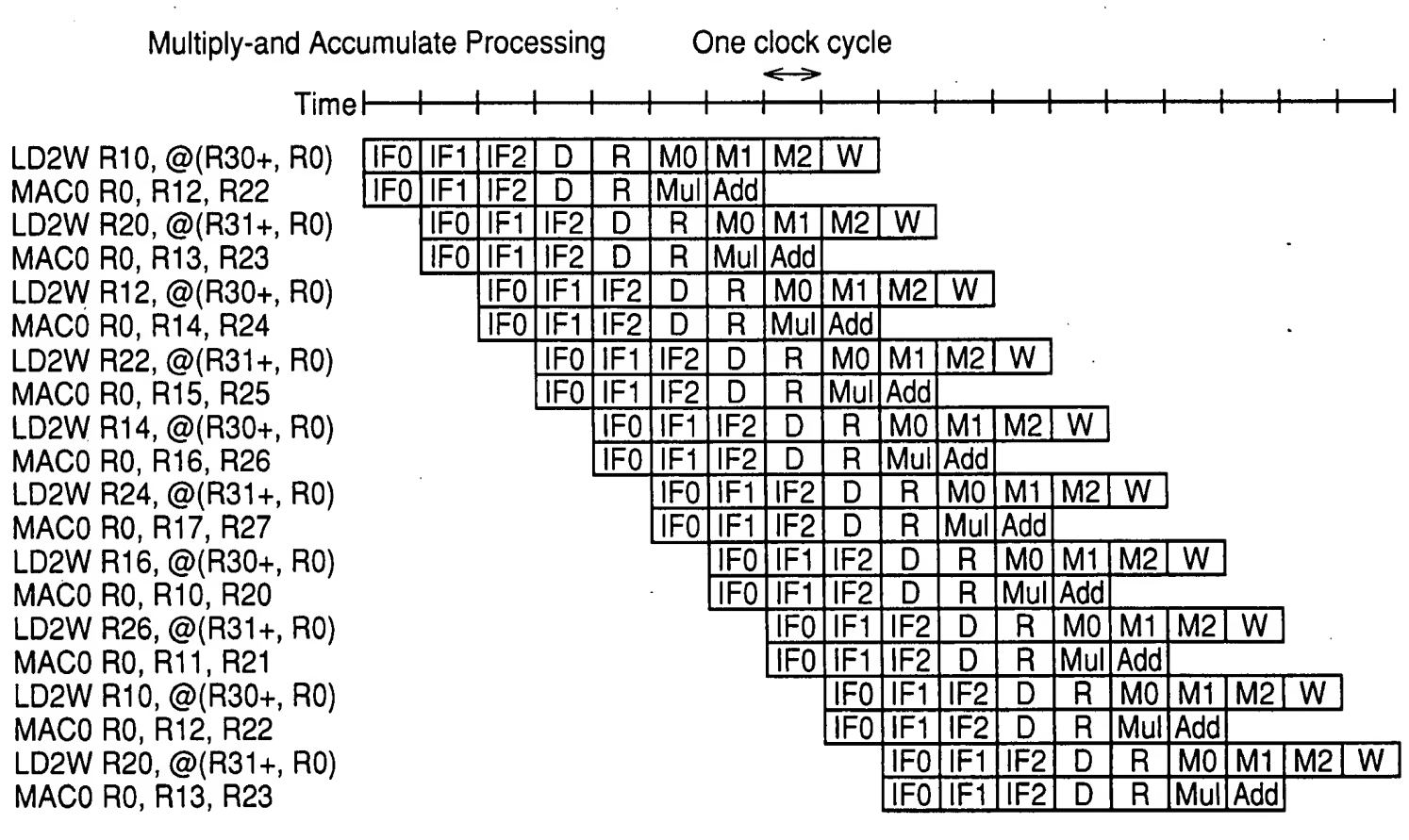


FIG.24